A Survey of Low-Jitter Guaranteed-Rate Scheduling for Input-Queued Networks

Prof. T.H. Szymanski
McMaster University
April 30, 2008
Talk Outline

- IP Networks - A Mesh of Routers
- Basic Switching: Input-Queueing vs Output-Queueing
- IQ Switch Scheduling = Bipartite Graph matching, MSM, MWM
- Birkoff-von-Neumann Stochastic Matrix Decomposition
- Convex Stochastic Matrix Decomposition
- Greedy Low-Jitter Stochastic Matrix Decomposition
- Recursive Fair Stochastic Matrix Decomposition
- Summary
Motivation

- **Low-Jitter scheduling algorithms** can provide many benefits, when incoming traffic is shaped:
  - Number of cells queued per switch very small (potentially 1-2 cells per flow, if jitter is low enough)
  - Queuing delay per switch very small (potentially 1-2 cell delays, if jitter is low enough)
  - Memory in switch reduced significantly, if jitter is low enough
  - End-to-End Queuing delay small (potentially 1-2 ideal cell delays per switch, if jitter per switch is low enough)
  - End-to-End Packet loss rate reduced to zero, if jitter is low enough
“The Internet is a Mesh of Routers”

Basic Switching Functions

IP SWITCHING

• Variable-size Incoming IP packets segmented into fixed-size cells at input-side of router

• cells transferred to output-side, where variable-size IP packets are re-assembled and forwarded

• Can be a complex process:
  • Where to store the cells: input-side, output-side, internally?
  • How to resolve contention for output ports?
  • How much ‘speedup’ (over-provisioning) is needed?
  • How to select cells for service to meet QoS constraints?
Basic Router

Leon-Garcia, Widjaja, Communication Networks

![Diagram of a Basic Router showing the internal structure including controller, line cards, interconnection fabric, and output ports.](image)
Basic Router Components

- **LINE CARDS**: receive IP packets, segment IP packets into cells, store cells, classify and police cells, schedule conforming cells through switch, re-assemble IP packets, forward IP packets, maintain QoS constraints

- **SWITCH**: transfer cells from input-side to output-side

- **CONTROL PROCESSORS / NETWORK PROCESSORS**: run switch operating system, run IP routing protocols, configure IP routing tables, run QoS protocols (RSVP, DiffServ), manage system

- **CELL FORWARDING ENGINES**: inspect IP packet headers, table lookup of destination switch port, re-write IP packet leaders
Moore’s Law: Link SpeedsGrowing Exponentially!


Packet processing Power

Link Speed

Moore’s law
2x / 18 months

2x / 7 months

Fiber Capacity (Gbit/s)


TDM DWDM

Source: SPEC95Int & David Miller, Stanford.

Low-Jitter Scheduling April 08 - pg 8

Prof. T.H. Szymanski, McMaster U
Scheduling Times Decreasing Exponentially!

- Try to do more (QoS) in less time!

<table>
<thead>
<tr>
<th>Year</th>
<th>Optical Link Rate</th>
<th>Time-Slot (512 bit cell)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990s</td>
<td>1 Gb/s</td>
<td>512 nsec</td>
</tr>
<tr>
<td>1995s</td>
<td>10 Gb/s</td>
<td>51.12 nsec</td>
</tr>
<tr>
<td>2000s</td>
<td>100 Gb/s</td>
<td>5.12 nsec</td>
</tr>
<tr>
<td>2007</td>
<td>1 Tb/s</td>
<td>0.512 nsec</td>
</tr>
</tbody>
</table>
Ideal OQ Switch

Input Ports

Switch Fabric

Output Ports

No Conflicts

No Conflicts (Speedup = N)

Output Link Scheduler (GPS / WFQ)
Input-Queued Switch

Input Ports

Switch Fabric

Output Ports

Potential Conflicts

Potential Conflicts

VOQ(0,0)

VOQ(0,N-1)

IP Packet Segmentation
Virtual Output Queues (VOQs)

Scheduler

IP Packet Reassembly
Output Link Scheduler
(GPS/WFQ)
IQ vs OQ Performance

Switch Architectures

- **OUTPUT-QUEUED CROSSBAR**: Ideal Performance - 100%, requires speedup = N, cost = $O(N^3)$ VLSI area, no scheduling at input-side, scheduling at output-side,

- **INPUT-QUEUED CROSSBAR**: Head-of-Line blocking - 58%, speedup = 1, cost = $O(N^2)$ VLSI area, complex scheduling at input-side, scheduling at output-side,

- **COMBINED INPUT-OUTPUT QUEUING**: can achieve 100 % throughput, requires speedup = 2..4, cost = $O(SN^2)$ VLSI area (S = speedup), complex scheduling at input-side, scheduling at output-side,
Switch Architectures

- **INTERNALLY-BUFFERED CROSSBAR**: Ideal Performance - 100%, speedup = 1, cost = $O(N^2)$ VLSI area + $O(N^2)$ internal cell buffers, scheduling at input-side, scheduling at output-side

- **MULTISTAGE SWITCHES**: 3-stage space-division-switches (REARRANGEABLE CLOS network), cost = $O(N^2)$ VLSI area, complex scheduling at input-side, complex routing required, scheduling at output-side,
Dynamic Scheduling - IQ Switches

- Equivalent to **Matching Problem** in a bipartite graph
- **Vertices** = input + output ports
- **Weighted edges** \((j,k)\) = \# cells in VOQ\((j,k)\)
- Find the matching = set of up to N matches (permutation) between \((I,O)\) pairs
- use permutation to configure crossbar switch for one time-slot
- Repeat matching problem for next time-slot
Dynamic Scheduling - IQ Switches


- Matching Algorithms:
  - **Maximal Matching (greedy) (MM):** $O(N^2)$ work, may not saturate
  - **Maximum Size Matching (MSM):** $O(N^3)$ work, will saturate if possible
  - **Maximum Weight Matching (MWM):** $O(N^3)$ work, will maximize the sum of edge weights in the matching
  - by selecting edge weights appropriately, many important aspects can be optimized
  - Weights = # cells in each VOQ, age of oldest cell in each VOQ, lag of the oldest in in each VOQ, relative to an ideal OQ switch, etc
Bipartite Graph Matching

• a matching is a partial or full permutation of (I,O) pairs, ie (2, 4, 3, 1)
Guaranteed-Rate Scheduling


• Compute a ‘scheduling frame’ consisting of many bipartite matchings to be used over a duration of time
• Every (I,O) pair specifies a Guaranteed-Rate of traffic for its QoS
• Traffic rates form an NxN traffic rate matrix
• An ‘admissible’ traffic rate matrix: no I is overloaded, no O is overloaded
• Compute all bipartite matchings such that every (I,O) pair is guaranteed to receive its requested service rate, over the duration of the scheduling frame
• No need to compute matchings on a one-by-one basis, in real-time
• But, we need to know the traffic rate matrix in advance
1999: NTH University Taiwan propose a BVN scheme which decomposes a traffic matrix into a sequence of switch configurations (permutations), using a classic theorem due to Birkoff and Von Neuman.

Requires large computation time $O(N^{4.5})$, achieves 100% throughput, speedup = 1 (optimal), but can have poor jitter $O(N^2)$ time; with scheduling $O(N^3\log N)$, service lag can be bounded.

2003: Stanford University & Bell Labs formulate NP-Hard low jitter decomposition problem; Propose Greedy approximation: relatively fast $O(N^3)$, can achieve 80% throughput, low jitter $O(1)$ time, requires speedup = $O(\log N)$, low efficiency.

2004: MIT proposes a ‘low-jitter’ scheme, bounds jitter to $O(N^1)$ time, speedup approaches 2, can achieve 100% throughput, speedup required between 1 and 2.

2006: McMaster proposes ‘low-jitter’ scheme, bounds maximum service lag to a small number of ‘IIDTs’, complexity $O(N F \log N F)$, achieves 100% throughput with speedup = 1.
Examples of low-jitter GR scheduling algorithms will be summarized next, followed by a closer look at BVN decomposition:

- BVN decomposition #1
- BVN decomposition #2
- Convex decomposition #1
- Greedy Low-Jitter decomposition #1

\[
R = \begin{bmatrix}
0.38 & 0 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22 \\
\end{bmatrix}
\]
Birkhoff-von-Neuman Decomposition #2

Input Ports

Switch Fabric

Output Ports

IP Packet Segmentation

Virtual Output Queues (VOQs)

Scheduler

IP Packet Reassembly

Output Link Scheduler

(GPS/WFDQ)

\[
R = \begin{bmatrix}
0.38 & 0 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22
\end{bmatrix}
\]

\[
R = (0.14) \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + (0.23) \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{bmatrix} + (0.37) \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} + (0.08) \times \begin{bmatrix}
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + (0.05) \times \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} + (0.09) \times \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix} + (0.03) \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} + (0.01) \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\]

• BVN decompositions are non-unique
Service Lead-Lag Plots


Rate Quantization

Koksal, Gallager, Rohrs,'Rate Quantization and Service Quality over Single Crossbar Switches', IEEE Infocom, 2004

- MIT approach: quantize guaranteed rates and relax bandwidth constraint

Fig. 3. Extreme points are expanded by a factor of $S$. The convex combination the expanded extreme points is a superset of the set of doubly stochastic matrices which constitute the set of admissible rates. Note that the set of admissible rates is unchanged.
Convex Decomposition (#3)

- Relax bandwidth constraint: let speedup > 1
- Bandwidth requirement = (0.60+0.38+0.23+0.22+0.05) = 1.48

\[
R \leq \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix} + 0.38 \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{bmatrix} + 0.23 \times \begin{bmatrix}
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix} + 0.22 \times \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} + 0.05 \times \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
• Relax bandwidth constraint, let speedup > 1

• **Bandwidth requirement** = (0.60+0.38+0.23+0.22+0.05) = 1.48

• Before speedup, increase in bandwidth requirement will increase the lag!

• In example #3, bandwidth requirement is 1.48, to meet GRs

• Before speedup, for some unlucky flows it takes 1.48 as much time to realize the required service => large service lags

• Before speedup, some flows may receive more service (1.48 times as much) than required => large service leads

• After speedup, all flows receive **at least** the required service
Integer Programming for Low-Jitter Decomposition


- formulate low-jitter scheduling of matrix R as an optimization problem
- given N! permutations, find a minimal set of permutations \( P_k \) and weights \( \phi_k \) such that \( R \leq \sum_{k=1}^{K} \phi_k \cdot P_k \), given constraints:
  - the guaranteed-rate between each (I,O) pair is due to one permutation (not spread over many permutations)
  - the number of permutations used is minimal
  - the sum of the permutation weights is minimal
  - required speedup given by sum of permutation weights
  - problem is NP-hard, so they proposed a greedy decomposition
  - worst-case speedup = \( O(\log N) \)
Greedy Low-Jitter Decomposition


\[
R \leq 0.60 \times \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} + 0.05 \times \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} + 0.33 \times \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} + 0.38 \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]

- Bandwidth requirement = \( (0.60 + 0.05 + 0.33 + 0.38) = 1.36 \)
• **BVN Decomposition** can be used to provide ‘Guaranteed Rate’ communications for all (I,O) pairs of an Input-Queued crossbar switch

• Based upon algorithms by Birkhoff and von Neumann on the decomposition of a doubly stochastic matrix

• Any doubly stochastic matrix can be decomposed exactly into a weighted sum of permutation matrices

• An IQ crossbar switch can then be scheduled by (a) decomposing a doubly stochastic traffic rate matrix, and (b) scheduling the resulting permutation matrices to appear in accordance with their respective weights

• Can achieve 100% throughput, ie every (I,O) pair is allocated exactly its guaranteed rate

• Requires only unity speedup
BVN Decomposition - 3 Key Algorithms


- **Algorithm #1 (von-Neumann, 1953):** Convert any doubly-sub-stochastic matrix into a doubly stochastic matrix

- **Algorithm #2 (Birkhoff - 1948):** Decompose any doubly stochastic matrix into a sum of weighted permutation matrices

- **Algorithm #3 (Chang et al - 1999):** Schedule weighted permutation matrices: Use “Weighted Fair Queueing” (WFQ) or “Generalized Processor Sharing” (GPS), algorithms developed at MIT, to schedule the matrices from the decomposition.
Let $R=(r_{i,j})$ be the doubly sub-stochastic Guaranteed-Rate traffic matrix.

**Birkhoff’s Theorem**: Given a doubly stochastic traffic rate matrix (that no IP or OP is overloaded):

- There exists a set of positive numbers (weights) $\phi_k$ and permutation matrices $P_k$, $k = 1,...,K$ for some $0 \leq K \leq N^2-2N+2$ that satisfy these two equations:

$$R \leq \sum_{k=1}^{K} \phi_k \cdot P_k,$$

where $\sum_{k=1}^{K} \phi_k = 1$

- 1st eq: The matrix $R$ is less than or equal to a weighted sum of permutation matrices

- 2nd eq: Sum of weights = 1.0 exactly
The number of permutation matrices = \( K \leq (N^2 - 2N + 2) = O(N^2) \)

The amount of work to find each permutation matrix is \( O(N^2) \), since any algorithm must search \( N^2 \) matrix elements.

Total work to find a decomposition is \( O(N^2) \) permutations * \( O(N^2) \) work per permutation = \( O(N^4) \)

Drawbacks:

Need to know the Guaranteed Rate traffic matrix in advance

Amount of work is excessive, \( O(N^4) \) computations, which is larger than the \( O(N^3) \) work in finding a Maximum Weight Matching
von Neumann’s Algorithm - #1

• **(von Neumann) Theorem:** If a matrix $R$ is doubly sub-stochastic, then there exists a doubly stochastic matrix $\tilde{R}$ such that every element in $R$ <= corresponding element in $\tilde{R}$

$$R = (r_{i,j}), \quad \tilde{R} = (\tilde{r}_{i,j}), \quad r_{i,j} \leq \tilde{r}_{i,j}, \forall i, j$$

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,N} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{N,1} & r_{N,2} & \cdots & r_{N,N} \end{pmatrix}$$

$$\tilde{R} = \begin{pmatrix} \tilde{r}_{1,1} & \tilde{r}_{1,2} & \cdots & \tilde{r}_{1,N} \\ \tilde{r}_{2,1} & \tilde{r}_{2,2} & \cdots & \tilde{r}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{r}_{N,1} & \tilde{r}_{N,2} & \cdots & \tilde{r}_{N,N} \end{pmatrix}$$

$$\sum_i r_{i,j} \leq 1, \sum_j r_{i,j} \leq 1$$

$$\sum_i \tilde{r}_{i,j} = 1, \sum_j \tilde{r}_{i,j} = 1$$
Algorithm 1

- **Step 1**: If the sum of all elements in R < N, then there exists an element \( r(i,j) \) such that: row_sum <1.0 and column_sum <1.0

- Find the element \( r(i,j) \), and compute \( \epsilon = 1 - \max[\text{row}_\text{sum}, \text{column}_\text{sum}] \)

- **Step 2**: Add \( \epsilon \) to the element \( r(i,j) \), which causes either the row-sum to equal 1.0 exactly, or the column sum to equal 1.0 exactly

- The number of rows plus columns whose sum is < 1.0 decreases by 1

\[
\text{if } \left( \sum_i r_{i,j} < 1 \right) \& \left( \sum_j r_{i,j} < 1 \right) \\
\text{then } \quad \epsilon = 1 - \max \left[ \sum_j r_{i,j}, \sum_i r_{i,j} \right] \\
\quad r_{i,j} = r_{i,j} + \epsilon
\]
Algorithm 1

- **Step-3**: Repeat these steps until the sum (all elements) = \( N \) exactly. In this case, the final matrix is **doubly stochastic**.

- The initial **doubly sub-stochastic** \( N \times N \) matrix has \( N \) rows and \( N \) columns.

- At most \( 2N \) rows and columns whose sum is < 1.0 exactly.

- Each iteration of (step 1 + step 2) decreases the number of rows and columns with sum < 1.0 by 1

- The algorithm terminates after at most \( 2N-1 \) iterations.

- Each iteration of (step1+step2) requires \( O(N^2) \) work

- The computational complexity: \( O(N^2) \) iterations * \( O(N^2) \) computations per iteration = \( O(N^4) \) computations
Algorithm 1 - Example

Initial Matrix: 
\[ R = \begin{bmatrix} 0.10 & 0 & 0.13 & 0.40 \\ 0.11 & 0.07 & 0.60 & 0.05 \\ 0 & 0.32 & 0.12 & 0.23 \\ 0.51 & 0.23 & 0.04 & 0.22 \end{bmatrix} \]

Final Matrix: 
\[ R = \begin{bmatrix} 0.38 & 0 & 0.22 & 0.40 \\ 0.11 & 0.24 & 0.60 & 0.05 \\ 0 & 0.53 & 0.14 & 0.33 \\ 0.51 & 0.23 & 0.04 & 0.22 \end{bmatrix} \]

STEP 1: 
\[ R = \begin{bmatrix} 0.10 & 0 & 0.13 & 0.40 \\ 0.11 & 0.07 & 0.60 & 0.05 \\ 0 & 0.32 & 0.12 & 0.23 \\ 0.51 & 0.23 & 0.04 & 0.22 \end{bmatrix} \]

\[ 0.72 \]

STEP 2: 
\[ R = \begin{bmatrix} 0.38 & 0 & 0.13 & 0.40 \\ 0.11 & 0.07 & 0.60 & 0.05 \\ 0 & 0.32 & 0.12 & 0.23 \\ 0.51 & 0.23 & 0.04 & 0.22 \end{bmatrix} \]

\[ 0.76 \]
• **Step 1**: given a doubly stochastic matrix \( \tilde{R} = (\tilde{r}_{ij}) \)

• Let \( (i_1, i_2, \ldots, i_N) \) be a permutation of \((1, 2, \ldots N)\) such that

\[
\prod_{k=1}^{N} \tilde{r}_{i_k, i_k} > 0
\]

• Example:

\( (i_1, i_2, \ldots, i_N) = (3, 4, 2, 1) \)

\[
R = \begin{bmatrix}
0.38 & 0 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22
\end{bmatrix}
\]

• **Step 2**: Let \( P_1 \) be the permutation matrix corresponding to \( (i_1, i_2, \ldots, i_N) \)

• and define \( \phi_1 = \min_{1 \leq k \leq N} (\tilde{r}_{k,i_k}) \) and \( R_1 = R - \phi_1 \cdot P_1 \)
Algorithm 2

- **Step 3**: If $R_1 = 0$ then the decomposition is complete

- **Step 4**: Otherwise, if $\phi_1 < 0$ then

$$R_2 = \frac{1}{1 - \phi_1} R_1$$

is still doubly stochastic, and the decomposition can proceed from step 1

- **Iterate** steps 1-4 until termination
Algorithm 2 - Complexity


- Algorithm 2 will terminate after at most $N^2 - 2N + 2$ steps

- Step 1: Finding a permutation $(i_1, i_2, \ldots, i_N)$ such that $\prod_{k=1}^{N} \tilde{r}_{i_k} > 0$ is equivalent to finding a maximum size matching in a bipartite graph

- Complexity of step 1 is $O(N^3)$ using the alternative path algorithm

- Complexity of step 1 is $O(N^{2.5})$ using a network flow algorithm

- Overall complexity is $O(N^{4.5})$ time
Algorithm 2 - Example


\[
R = \begin{bmatrix}
0.38 & 0 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22
\end{bmatrix}
\]

\[
R = 0.14 \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + 0.23 \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{bmatrix} + 0.10 \times \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix} + 0.01 \times \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + 0.36 \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} + 0.04 \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix} + 0.07 \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + 0.05 \times \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]
From Algorithms 1 & 2, we can find a set of positive numbers (weights) $\phi_k$ and permutation matrices $P_k$, $k = 1, \ldots, K$ for some $K \leq N^2 - 2N + 2$ that satisfies the 2 equations:

$$R \leq \sum_{k=1}^{K} \phi_k P_k ,$$

$$\sum_{k=1}^{K} \phi_k = 1.$$

Given the permutations and weights, we can schedule the permutations to realize the traffic requirements of the Guaranteed-Rate traffic matrix $R$.

Candidate scheduling algorithm: Packetized Generalized Processor Sharing (PGPS) algorithm or WFQ algorithm, developed at MIT.
Given the permutations and weights, schedule the crossbar switch to realize the traffic requirements of the Guaranteed-Rate traffic matrix R.

- **Step 1**: Set discrete time-slot $n=1$ initially.
- Given each permutation $k$ a Finishing Time $F(k, 1) = \frac{1}{\phi_k}$
- Sort the $k$ tokens in order of increasing finishing times.
- **Step 2**: Select the smallest token, $F(k, c)$, schedule permutation $k$ for time-slot $n$, and increment time $n$.
- Step 3: Assign the token a new Finishing Time $F(k, c + 1) = F(k, c) + \frac{1}{\phi_k}$ and insert the token into the sorted list

- **Repeat** Steps 2,3
There are exactly \( K \) tokens in the sorted token list, one for each permutation matrix.

Each token has a finishing time. Given \( K \) tokens in the sorted list at any one time, where \( K = N^2 - 2N + 2 \), the computational complexity of inserting one token into the list is \( O(\log N^2) = O(\log N) \).

The Computational Complexity of Algorithm 3 is \( O(N^3 \log N) \) (see the paper).

In an $N \times N$ IQ switch, let the traffic rate between IO pair $(j,k)$ be denoted: $r_{j,k}$

Let the cumulative service of the traffic between IO pair $(j,k)$ by time $t$ be denoted: $C_{j,k}(t)$ (measured in time-slots)

Claim: Given an admissible traffic rate matrix $r$, than there is a scheduling algorithm such that:

$$C_{j,k}(t) - C_{j,k}(s) \geq r_{j,k}(t-s) - S_{j,k}$$

for all input/output ports $j,k$, and for all times $s \leq t$, and for some service lag $S_{j,k} \leq N^2 - 2N + 2$
Ideally: Cumulative service (at time t) = Cumulative service (at time s) + (Guaranteed Service Rate \( r_{j,k} \)) * (interval t-s)

BVN: Cumulative service (at time t) = Cumulative service (at time s) + (Guaranteed Service Rate \( r_{j,k} \)) * (interval t-s) - ‘service lag’ term

\( r_{j,k} \) can be expressed in time-slots of service per second, \( S_{j,k} = \) ‘service lag’ in time-slots
Given any admissible rate matrix $R=(r_{i,j})$, algorithms 1,2, and 3 generate a scheduling policy that guarantees fair service and bounded service lag:

$$C_{i,j}(t) - C_{i,j}(s) \geq r_{i,j}(t-s) - S_{i,j}$$

For all (I,O) pairs $i,j$, for all times $s \leq t$, and some service lag $S_{i,j} \leq N^2 - 2N +2$. 
Example from the paper:


\[
R = \begin{bmatrix}
0.10 & 0 & 0.13 & 0.40 \\
0.11 & 0.07 & 0.60 & 0.05 \\
0 & 0.32 & 0.12 & 0.23 \\
0.51 & 0.23 & 0.04 & 0.12
\end{bmatrix}.
\]

\[
\tilde{R} = 0.14 \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} + 0.04 \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\tilde{R} = \begin{bmatrix}
0.38 & 0 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22
\end{bmatrix}.
\]

\[
\tilde{R} + 0.40 \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} + 0.08 \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
+ 0.01 \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} + 0.12 \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{bmatrix}
\]

\[
S = \begin{bmatrix}
6.66 & 0 & 4.54 & 3.8 \\
1.77 & 3.68 & 7.2 & 2.35 \\
0 & 7.71 & 1.98 & 5.31 \\
6.57 & 3.61 & 1.28 & 3.54
\end{bmatrix}.
\]

\[
S + 0.11 \begin{bmatrix}
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0
\end{bmatrix} + 0.10 \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix}.
\]
Proposed Recursive Fair Decomposition


• Given a traffic rate matrix $R$, create doubly-stochastic matrix $\tilde{R}$, and create quantized rate matrix $M$ (given a frame length $F$)

• Recursively decompose matrix $M$, to yield a sequence of permutation matrices

• Use these permutations to configure the IQ switch

\[
R = \begin{bmatrix}
0.10 & 0.00 & 0.13 & 0.40 \\
0.11 & 0.07 & 0.60 & 0.05 \\
0.00 & 0.32 & 0.12 & 0.23 \\
0.51 & 0.23 & 0.04 & 0.22 \\
\end{bmatrix}, \quad \tilde{R} = \begin{bmatrix}
0.38 & 0.00 & 0.22 & 0.40 \\
0.11 & 0.24 & 0.60 & 0.05 \\
0.00 & 0.53 & 0.14 & 0.33 \\
0.51 & 0.23 & 0.04 & 0.22 \\
\end{bmatrix}
\]
Proposed Recursive Fair Decomposition


“Method and Apparatus to Schedule Packets through a crossbar switch with Delay Guarantees,” US Patent App .07


\[
\begin{bmatrix}
0.3799 & 0 & 0.2197 & 0.4004 \\
0.1104 & 0.2393 & 0.5996 & 0.0498 \\
0 & 0.5303 & 0.1396 & 0.3291 \\
0.5098 & 0.2295 & 0.0400 & 0.2197
\end{bmatrix}
= \\
\begin{bmatrix}
0.1895 & 0 & 0.1104 & 0.2002 \\
0.0557 & 0.1191 & 0.2998 & 0.0244 \\
0 & 0.2656 & 0.0693 & 0.1650 \\
0.2549 & 0.1142 & 0.0205 & 0.1094
\end{bmatrix}
+ \\
\begin{bmatrix}
0.1904 & 0 & 0.1094 & 0.2002 \\
0.0547 & 0.1201 & 0.2998 & 0.0254 \\
0 & 0.2646 & 0.0703 & 0.1641 \\
0.2549 & 0.1152 & 0.0195 & 0.1104
\end{bmatrix}
\]

• first step of decomposition: guaranteed rates split fairly evenly over two resulting doubly-stochastic matrices, ensuring low-jitter in each half of original frame
(a) Service Lead-Lag curve for 100 matrices, size 8x8, with 6,400 GR traffic flows, at 100% load, with unity speedup
End-to-End Path, 10 IP Routers

- Consider a linear chain of Input-Queued 8x8 routers, H=10 hops
- Let there be many distinct, competing, simultaneous Guaranteed-Rate traffic flows such that each switch load = 100%, at unity speedup
- schedule all switches, and observe the results for End-to-End flows
End-to-End Delay and Jitter


<table>
<thead>
<tr>
<th>IP Router #</th>
<th>E(Nq)</th>
<th>E(Wq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.625</td>
<td>0.477 us</td>
</tr>
<tr>
<td>2</td>
<td>1.961</td>
<td>1.496 us</td>
</tr>
<tr>
<td>3</td>
<td>1.086</td>
<td>0.829 us</td>
</tr>
<tr>
<td>4</td>
<td>1.748</td>
<td>1.334 us</td>
</tr>
<tr>
<td>5</td>
<td>2.822</td>
<td>2.153 us</td>
</tr>
<tr>
<td>6</td>
<td>1.287</td>
<td>0.982 us</td>
</tr>
<tr>
<td>7</td>
<td>1.369</td>
<td>1.045 us</td>
</tr>
<tr>
<td>8</td>
<td>1.664</td>
<td>1.270 us</td>
</tr>
<tr>
<td>9</td>
<td>1.406</td>
<td>1.073 us</td>
</tr>
<tr>
<td>10</td>
<td>0.953</td>
<td>0.727 us</td>
</tr>
<tr>
<td>Playback Q</td>
<td>0.4531</td>
<td>0.298 us</td>
</tr>
</tbody>
</table>

Low jitter per switch

Zero jitter after small playback buffer

Small queues
Low-Jitter Scheduling, Silicon Fat-Trees


- Consider 256 competing simultaneous GR traffic flows, routed through a 2-level Fat-Tree with 16 crossbar switches, at 100% capacity, with unity speedup. Low End-to-End delay, with essentially-zero End-to-End delay jitter (after small playback buffer).
Low-Jitter Scheduling, Silicon Fat-Trees


Low jitter per switch

Small queues

FIG 5(a) IAT PDF, 1-hop path. (b) Queue occupancy, 1-hop path.

FIG 6(a) IAT PDF, 3-hop path. 6(b) Queue occupancy, 3-hop path.
Conclusions

- BVN decomposition and scheduling for IQ switches, 100% efficiency at unity speedup, jitter bound $O(N^2)$, time complexity $O(N^{4.5})$

- Convex decomposition and scheduling for IQ switches, 100% logical efficiency at speedup approaching 2 (=> approx 50% real efficiency), jitter bound $O(N)$ at speedup = 2, time complexity $O(N^3)$ (tbc)

- Greedy Low-Jitter decomposition and scheduling for IQ switches, 100% logical efficiency at speedup approaching 2 (=> approx 50% real efficiency), worst-case speedup = $O(\log N)$, jitter bound $O(1)$ at speedup = $O(\log N)$, greedy time complexity $O(N^3)$ (tbc)

- Recursive fair decomposition and scheduling for IQ switches, 100% efficiency of IO ports at speedup =1, jitter bound $O(K*\text{IIDT})$ for constant $K$, time complexity $O(NF \log(NF))$
Thank you.

Questions ?